Name

intro – introduction to network library functions

Description

This section describes functions that are available for interprocess communication (IPC). IPC takes place using sockets. The `socket`(2) system call creates a communications channel based on domain, type, and protocol.

Sockets are created without names. The `bind`(2) system call is used to connect a name to a socket.

A connection with another process must be made before data can be transferred on a bound socket. The `connect`(2) system call is used to rendezvous with another process. This process must be listening on a bound socket using the `listen`(2) system call. This listening process can accept a connection request using the `accept`(2) system call.

Once two processes have connected and accepted an IPC, data can be transferred with the following system calls: `read`(2); `write`(2); `send`(2), and `recv`(2).

Connectionless sockets are also possible (a socket is bound and data can be transferred). They use the following system calls to transfer data: `sendto` and `recvfrom`.

IPC operates in three domains:

UNIX	Local node
INTERNET	Local area network (LAN)
DECNET	DECnet network

These types of sockets are available for IPC:

stream	Sequenced, reliable, unduplicated data CONNECTED socket record boundaries not preserved all domains
datagram	Not guaranteed to be sequenced, reliable, or unduplicated user protocol needed to give guarantees UNCONNECTED socket record boundaries preserved UNIX and INTERNET domains
sequenced packet	Like stream socket, except record boundaries preserved DECNET domain only
raw	Access to communications protocols

Internet Addresses Routines

The *inet* routines manipulate Internet addresses.

Network Data Base File Routines

Standard mapping routines are used to retrieve entries in network data base files. Several routines operating on each data base file are identified by a group name:

`gethostent`	Retrieves entries from `/etc/hosts`
`getnetent`	Retrieves entries from `/etc/networks`
`getprotoent`	Retrieves entries from `/etc/protocols`
`getservent`	Retrieves entries from `/etc/services`

Specific routines perform particular operations on each data base file:

`get...ent`	Reads the next line of the file; opens the file, if necessary.
`set...ent`	Opens and rewinds the file.
`end...ent`	Closes the file.
`get...byname`	Searches the file sequentially from the beginning until a matching *name* is found, or EOF is encountered.
`get...byaddr`	Searches the file sequentially from the beginning until a matching *address* is found, or EOF is encountered.
`get...byport`	Searches the file sequentially from the beginning until a matching *port number* is found, or EOF is encountered.
`get...bynumber`	Searches the file sequentially from the beginning until a matching *protocol number* is found, or EOF is encountered.

Each network library routine returns a pointer to a structure reflecting individual fields of a line in one of the network data base files. The structure for each data base file contains some of the fields in the following list, with the prefix *x* replaced by a different letter in each file:

`x_addr`	pointer to a network address, returned in network-byte order
`x_addrtype`	address family of the address being returned
`x_aliases`	alternate names
`x_length`	length of an address, in bytes
`x_name`	official name
`x_net`	network number, returned in machine-byte order
`x_port`	resident port
`x_proto`	protocol number

Name

htonl, htons, ntohl, ntohs – convert values between host and network byte order

Syntax

```
#include <sys/types.h>
#include </bsd/netinet/in.h>

netlong = htonl(hostlong);
u_long netlong, hostlong;

netshort = htons(hostshort);
u_short netshort, hostshort;

hostlong = ntohl(netlong);
u_long hostlong, netlong;

hostshort = ntohs(netshort);
u_short hostshort, netshort;
```

Description

These routines convert 16 and 32 bit quantities between network byte order and host byte order. These routines are defined as null macros in the include file <netinet/in.h>.

These routines are most often used in conjunction with Internet addresses and ports as returned by `gethostbyname`(3n) and `getservent`(3n).

See Also

gethostbyname(3n), getservent(3n)

Name

htonl, htons, ntohl, ntohs – convert values between host and network byte order

Syntax

```
#include <sys/types.h>
#include <netinet/in.h>

netlong = htonl(hostlong);
u_long netlong, hostlong;

netshort = htons(hostshort);
u_short netshort, hostshort;

hostlong = ntohl(netlong);
u_long hostlong, netlong;

hostshort = ntohs(netshort);
u_short hostshort, netshort;
```

Description

These routines convert 16-bit and 32-bit quantities between network byte order and host byte order. On some non-ULTRIX machines these routines are defined as null macros in the include file <netinet/in.h>.

These routines are most often used with Internet addresses and ports as returned by `gethostent`(3n) and `getservent`(3n).

Restrictions

The VAX handles bytes in the reverse from most everyone else.

See Also

gethostent(3n), getservent(3n)

Name

gethostent, gethostbyaddr, gethostbyname, sethostent, endhostent – get hosts entry

Syntax

#include <netdb.h>

struct hostent *gethostent()

struct hostent *gethostbyname(*name***)**
char ***name*;**

struct hostent *gethostbyaddr(*addr***, *len*, *type*)**
char ***addr*; int *len*, *type*;**

sethostent(*stayopen***)**
int *stayopen*;

endhostent()

Description

The `gethostent`, `gethostbyname`, and `gethostbyaddr` subroutines return a pointer to an object with the following structure containing the broken-out fields reflecting information obtained from the `hosts` database.

```
struct   hostent {
         char   *h_name;       /* official name of host */
         char   **h_aliases;   /* alias list */
         int    h_addrtype;    /* address type */
         int    h_length;      /* length of address */
         char   **h_addr_list; /* list of addresses from name server */
#define  h_addr h_addr_list[0] /* address for backward compatibility */
};
```

The members of this structure are:

h_name	Official name of the host.
h_aliases	A zero terminated array of alternate names for the host.
h_addrtype	The type of address being returned; currently always AF_INET.
h_length	The length, in bytes, of the address.
h_addr	A pointer to the network address for the host. Host addresses are returned in network byte order.

If the *stayopen* flag on a `sethostent` subroutine is NULL, the hosts database is opened. Otherwise the `sethostent` has the effect of rewinding the `hosts` database. The `endhostent` may be called to close the `hosts` database when processing is complete.

The `gethostent` subroutine simply reads the next line while `gethostbyname` and `gethostbyaddr` search until a matching *name,* or *addr, len, type* is found (or until EOF is encountered). The `gethostent` subroutine keeps a pointer in the database, allowing successive calls to be used to search the entire file.

The gethostbyname and gethostbyaddr subroutines query the hosts database.

A call to sethostent must be made before a while loop using gethostent in order to perform initialization and an endhostent must be used after the loop. Both gethostbyname and gethostbyaddr make calls to sethostent and endhostent.

Restrictions

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.

If YP is running, gethostent does not return the entries in any particular order. See the *Guide to the Yellow Pages Service* for setup information.

The hosts database may also be distributed via the BIND/Hesiod naming service. See the *Guide to the BIND/Hesiod Service* for more information.

Return Value

Null pointer (0) returned on EOF or error.

Files

/etc/hosts

See Also

hosts(5), svc.conf(5)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent – get networks entry

Syntax

#include <netdb.h>

struct netent *getnetent()

struct netent *getnetbyname(*name***)**
char ***name*;**

struct netent *getnetbyaddr(*net***, *type*)**
long *net*; int *type*;

setnetent(*stayopen***)**
int *stayopen*;

endnetent()

Description

The `getnetent`, `getnetbyname`, and `getnetbyaddr` subroutines each return a pointer to an object with the following structure containing the broken-out fields of a line in the `networks` database.

```
struct netent {
        char    *n_name;      /* official name of net */
        char    **n_aliases;  /* alias list */
        int     n_addrtype;   /* net number type */
        long    n_net;        /* net number */
};
```

The members of this structure are:

n_name	The official name of the network.
n_aliases	A zero terminated list of alternate names for the network.
n_addrtype	The type of the network number returned: AF_INET.
n_net	The network number. Network numbers are returned in machine byte order.

If the *stayopen* flag on a `setnetent` subroutine is NULL, the `networks` database is opened. Otherwise the `setnetent` has the effect of rewinding the `networks` database. The `endnetent` may be called to close the `networks` database when processing is complete.

The `getnetent` subroutine simply reads the next line while `getnetbyname` and `getnetbyaddr` search until a matching *name* or *net* number is found (or until EOF is encountered). The *type* must be AF_INET. The `getnetent` subroutine keeps a pointer in the database, allowing successive calls to be used to search the entire file.

A call to `setnetent` must be made before a `while` loop using `getnetent` in order to perform initialization and an `endnetent` must be used after the loop. Both `getnetbyname` and `getnetbyaddr` make calls to `setnetent` and `endnetent`.

Restrictions

All information is contained in a static area so it must be copied if it is to be saved. Only Internet network numbers are currently understood.

If YP is running, `getnetent` does not return the entries in any particular order. See the *Guide to the Yellow Pages Service* for setup information.

The networks database may also be distributed via the BIND/Hesiod naming service. See the *Guide to the BIND/Hesiod Service* for more information.

Return Value

Null pointer (0) returned on EOF or error.

Files

`/etc/networks`

See Also

networks(5), svc.conf(5)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent – get protocols entry

Syntax

#include <netdb.h>

struct protoent *getprotoent()

struct protoent *getprotobyname(*name***)**
char ***name*;**

struct protoent *getprotobynumber(*proto***)**
int *proto***;**

setprotoent(*stayopen***)**
int *stayopen***;**

endprotoent()

Description

The `getprotoent`, `getprotobyname`, and `getprotobynumber` subroutines each return a pointer to an object with the following structure containing the broken-out fields of a line in the `protocols` database.

```
struct protoent {
        char   *p_name;      /* official name of protocol */
        char   **p_aliases;  /* alias list */
        long   p_proto;      /* protocol number */
};
```

The members of this structure are:

p_name The official name of the protocol.

p_aliases A zero terminated list of alternate names for the protocol.

p_proto The protocol number.

If the *stayopen* flag on a `setprotoent` subroutine is NULL, the protocols database is opened. Otherwise the `setprotoent` has the effect of rewinding the protocols database. The `endprotoent` may be called to close the protocols database when processing is complete.

The `getprotoent` subroutine simply reads the next line while `getprotobyname` and `getprotobynumber` search until a matching *name* or *proto* number is found (or until EOF is encountered). The `getprotoent` subroutine keeps a pointer in the database, allowing successive calls to be used to search the entire file.

A call to `setprotoent` must be made before a `while` loop using `getprotoent` in order to perform initialization and an `endprotoent` must be used after the loop. Both `getprotobyname` and `getprotobynumber` make calls to `setprotoent` and `endprotoent`.

Restrictions

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.

If YP is running, `getprotoent` does not return the entries in any particular order. See the *Guide to the Yellow Pages Service* for setup information.

The services database may also be distributed using the BIND/Hesiod naming service. See the *Guide to the BIND/Hesiod Service* for more information.

Return Value

Null pointer (0) returned on EOF or error.

Files

`/etc/protocols`

See Also

protocols(5), svc.conf(5)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

getservent, getservbyname, getservbyport, setservent, endservent – get services entry

Syntax

#include <netdb.h>

struct servent *getservent()

struct servent *getservbyname*(*name, proto*)
char **name*, **proto*;

struct servent *getservbyport*(*port, proto*)
int *port*; **char** **proto*;

setservent(*stayopen*)
int *stayopen*

endservent()

Description

The `getservent`, `getservbyname`, and `getservbyport` subroutines each return a pointer to an object with the following structure containing the broken-out fields of a line in the network `services` database.

```
struct servent {
        char    *s_name;        /* official name of service */
        char    **s_aliases;    /* alias list */
        long    s_port;         /* port service resides at */
        char    *s_proto;       /* protocol to use */
};
```

The members of this structure are:

s_name — The official name of the service.

s_aliases — A zero terminated list of alternate names for the service.

s_port — The port number at which the service resides. Port numbers are returned in network byte order.

s_proto — The name of the protocol to use when contacting the service.

If the *stayopen* flag on a `setservent` subroutine is NULL, the `services` database is opened. Otherwise, the `setservent` has the effect of rewinding the `services` database. The `endservent` subroutine may be called to close the `services` database when processing is complete.

The `getservent` subroutine reads the next line; `getservbyname` and `getservbyport` search until a matching *name* or *port* is found (or until EOF is encountered). The `getservent` subroutine keeps a pointer in the database, allowing successive calls to be used to search the entire file. If a non-NULL protocol name, `proto`, is also supplied, searches must also match the protocol.

The `setservent` routine must be called before a `while` loop that uses `getservent` in order to initialize variables in the `setservent` routine and an `endservent` must be used after the loop. Both `getservbyport` and `getservbyname` make calls to `setservent` and `endservent`.

Restrictions

All information is contained in a static area so it must be copied if it is to be saved.

If the Yellow Pages Service is running, `getservent` does not return the entries in any particular order. See the *Guide to the Yellow Pages Service* for setup information.

The `services` database can also be distributed by the BIND/Hesiod naming service. See the *Guide to the BIND/Hesiod Service* for more information.

Return Value

Null pointer (0) returned on EOF or error.

Files

`/etc/services`

See Also

services(5), svc.conf(5)
Guide to the BIND/Hesiod Service
Guide to the Yellow Pages Service

Name

inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof – Internet address manipulation routines

Syntax

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long inet_addr(cp)
char *cp;

unsigned long inet_network(cp)
char *cp;

char *inet_ntoa(in)
struct in_addr in;

struct in_addr inet_makeaddr(net, lna)
int net, lna;

int inet_lnaof(in)
struct in_addr in;

int inet_netof(in)
struct in_addr in;
```

Description

The routines `inet_addr` and `inet_network` each interpret character strings representing numbers expressed in the Internet standard ''.'' notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine `inet_ntoa` takes an Internet address and returns an ASCII string representing the address in ''.'' notation. The routine `inet_makeaddr` takes an Internet network number and a local network address and constructs an Internet address from it. The routines `inet_netof` and `inet_lnaof` break apart Internet host addresses, returning the network number and local network address part, respectively.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

Internet Addresses

Values specified using the ''.'' notation take one of the following forms:

```
a.b.c.d
a.b.c
a.b
a
```

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the VAX, the bytes referred to above appear as ''d.c.b.a''. That is, VAX bytes are ordered from right to left.

When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as ''128.net.host''.

When a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as ''net.host''.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as ''parts'' in a ''.'' notation may be decimal, octal, or hexadecimal, as specified in the C language (i.e. a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

Return Value

The value –1 is returned by `inet_addr` and `inet_network` for malformed requests.

See Also

gethostent(3n), getnetent(3n), hosts(5), networks(5)

Name

snmpextregister, snmpextgetreq, snmpextrespond, snmpexterror – library routines available for building the Extended ULTRIX SNMP Agent (Extended Agent)

Syntax

```
#include <protocols/snmp.h>
#include <protocols/snmperrs.h>

struct objident {
        short           ncmp;           /* number of components */
        unsigned long   cmp[SNMPMXID];  /* components */
};

struct snmpareg {
        short           oidtype;        /* object id type */
        objident        oid;            /* object id/*
};

struct snmparspdat {
        short           type;           /* response data type */
        short           octets;         /* number of octets in response data */
        char            *rspdat;        /* response data */
};

snmpextregister(reg, community)
struct snmpareg *reg;
char *community;

snmpextgetreq(reqoid, reqinst)
objident *reqoid;
objident *reqinst;

snmpextrespond(reqoid, rspinst, rspdat)
objident *reqoid;
objident *rspinst;
struct snmparspdat *rspdat;

snmpexterror(error)
long error;
```

Description

The following library routines are available for building the Extended Agent:

snmpextregister
Used to register the Extended Agent's Management Information Base (MIB) to the ULTRIX SNMP Agent (Agent). The *reg* parameter is provided by the caller with the object identifiers to be registered. The *community* parameter is provided by the caller with the community name (a null-terminated string).

This library routine waits for a registration confirmation from the Agent. The process is blocked until the confirmation arrives. When the confirmation arrives, the routine returns the status of the registration.

The program issues this call before any other Extended SNMP Library calls. It does this because the `snmpextregister` library routine creates a UNIX domain socket to the Agent on behalf of the caller.

snmpextgetreq

Used to receive a request for a MIB variable from the Agent. If there is no outstanding request from the Agent, the process is blocked until a request arrives from the Agent.

When the Extended Agent receives a request from the Agent, the *reqoid* parameter contains the object identifier for the requested variable. The *reqinst* parameter contains the object instance identifier for the requested variable. If the request does not contains an object instance, the *reqinst->ncmp* record contains a zero.

snmpextrespond

Used to return the requested variable to the Agent. The *reqoid* parameter is the object identifier from the `snmpextgetreq` library call. The *rspinst* parameter is the object instance associated with the returning variable. If there is no object instance associated with the returning variable, a null parameter must be supplied. The *rspdat* parameter is the returning variable.

Note that the Agent maintains a configurable timer for outstanding requests to the Extended Agent. Therefore, the Extended Agent must be able to respond within the Agent's timeout interval in order to prevent a premature timeout in the Agent.

See the `/etc/snmpd.conf` file for your system's default timeout value.

snmpexterror

Used to return an error to the Agent. The *error* parameter is the error code to be returned to the Agent. The error code is one of the following:

NOERR—successful SNMP *get-next-request end-of-table*. This happens when the requested instance does not exist.

NOSUCH—Unknown requested object identifier.

GENERRS—Generic error.

BADVAL—Bad variable value.

Restrictions

For the `snmpextregister` routine, the object identifier must have the prefix 1.3.6.1 to be registered. If it does not, the registration is rejected.

Return Value

If an error occurs, a negative value is returned.

Diagnostics

[BADVERSION]	Bad or obsolete protocol version
[BINDERR]	Failed to bind the socket
[GENSUC]	MIB successfully registered
[NOSOCK]	Socket does not exist
[NOSVC]	MIB registration was rejected
[PKTLENERR]	Maximum size message exceeded or community name is too large
[RCV_ERR]	Reception failed
[SND_ERR]	Transmission failed

Files

`/etc/snmpd.conf`	SNMP configuration file

See Also

snmpd.conf(5n), snmpd(8n), snmpsetup(8n)
Guide to Network Programming